Методы Оптимизации
Лабораторная работа №6

Вариант: №1
Выполнил: Бабушкин Александр P3221

Задание:

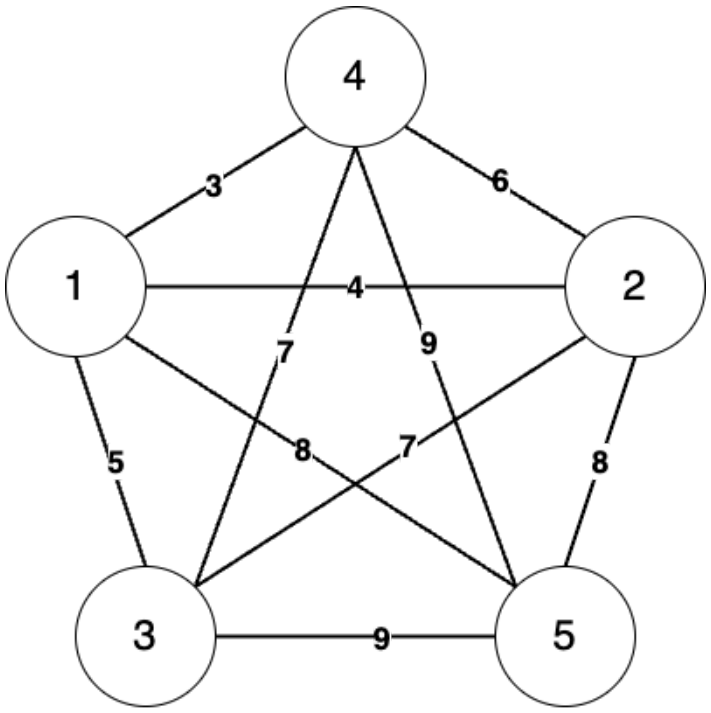
Дано множество из n городов и матрица расстояний между ними. Требуется объехать все города по кратчайшему пути, причем в каждом городе необходимо побывать один раз и вернуться в город, из которого был начат маршрут. Задачу необходимо решить с помощью генетического алгоритма.

	Город 1	**Город 2**	**Город 3**	**Город 4**	**Город 5**
Город 1	0	4	5	3	8
Город 2	4	0	7	6	8
Город 3	5	7	0	7	9
Город 4	3	6	7	0	9
Город 5	8	8	9	9	0

За целевую функцию следует принять сумму расстояний между городами.
Размер популяции N = 4.
Оператор мутации представляет собой случайную перестановку двух чисел в геноме, которые выбираются случайно. Вероятность мутации 0.01.

Решение:

Визуализация стоимости перемещения между городами:



Исходная популяция:

№	Последовательность	Функция
1	5 - 4 - 1 - 2 - 3	32
2	3 - 4 - 1 - 5 - 2	33
3	5 - 3 - 1 - 4 - 2	31
4	3 - 2 - 1 - 5 - 4	35

Ср. значение: $\frac{32 + 33 + 31 + 35}{4} = 32.75$

I. Первое поколение:

1) Для начала возьмем для скрещивания наборы последовательностей 3 и 4.

Возьмем первую границу после 1 числа, вторую после 3.

Получаем: |31| и |21|.

Произведем замену середин.

Вставляем числа из родительских последовательностей.

Для первого:

1 пропускаем, берем 4, 2 пропускаем, берем 5 и 3.

Получаем: 4 - 2 - 1 - 5 - 3 = 34

Для второго:

1 пропускаем, 5 и 4 берем, 3 пропускаем, 2 берем.

Получаем: 5 - 3 - 1 - 4 - 2 = 31

2) Теперь возьмем последовательности 1 и 2.

Возьмем границу после 2 и 4 числа.

Получаем: |12| и |15|.

Произведем замену середин.

Вставляем числа из родительских последовательностей.

Для первого:

Берем 3, 5 пропускаем, 4 берем, 1 пропускаем, 2 берем.

Получаем: 3 - 4 - 1 - 5 - 2 = 33

Для второго:

2 пропускаем, 3 и 4 берем, 1 пропускаем, 5 берем.

Получаем: 3 - 4 - 1 - 2 - 5 = 31

Проведем мутацию (если необходимо):

0.03, 0.53, 0.28, 0.78 — все значения больше порога мутации $\Rightarrow$ мутацию не проводим.

Получаем:

№	Родители	Потомки	Функция
3	5 - 3 - 1 - 4 - 2	4 - 2 - 1 - 5 - 3	31
4	3 - 2 - 1 - 5 - 4	5 - 3 - 1 - 4 - 2	34
1	5 - 4 - 1 - 2 - 3	3 - 4 - 1 - 5 - 2	31
2	3 - 4 - 1 - 5 - 3	3 - 4 - 1 - 5 - 2	33

Уберем худшие особи.

Текущая популяция:

№	Последовательность	Функция
1	5 - 4 - 1 - 2 - 3	32
2	3 - 4 - 1 - 5 - 2	33
3	5 - 3 - 1 - 4 - 2	31
4	3 - 4 - 1 - 2 - 5	31

Ср. значение: $\frac{32 + 33 + 31 + 31}{4} = 31.75$

II. Второе поколение:

1) Для начала возьмем для скрещивания наборы последовательностей 1 и 3.

Возьмем первую границу после 1 числа, вторую после 3.

Аналогично первому поколению найдем потомков.

Для первого:

1 пропускаем, 2 берем, 3 пропускаем, 5 и 4 берем.

Получаем: 2 - 3 - 1 - 5 - 4 = 35

Для второго:

1 и 4 пропускаем, 2, 5 и 3 берем.

Получаем: 2 - 4 - 1 - 5 - 3 = 33

2) Теперь возьмем последовательности 2 и 4.

Возьмем границу после 1 и 4 числа.

Получаем: |415| и |412|.

Аналогично первому поколению найдем потомков.

Для первого:

Получаем: 5 - 4 - 1 - 2 - 3 = 32

Для второго:

Получаем: 2 - 4 - 1 - 5 - 3 = 33

Проведем мутацию (если необходимо):

0.12, 0.02, 0.91, 0.15 — все значения больше порога мутации ⇒ мутацию не проводим.

Получаем:

№	**Родители**	**Потомки**	**Функция**
1	5 - 4 - 1 - 2 - 3	2 - 3 - 1 - 5 - 4	35
3	5 - 3 - 1 - 4 - 2	2 - 4 - 1 - 5 - 3	33
2	3 - 4 - 1 - 5 - 2	5 - 4 - 1 - 2 - 3	32
4	3 - 4 - 1 - 2 - 5	2 - 4 - 1 - 5 - 3	33

Уберем худшие особи.

Текущая популяция:

№	**Последовательность**	**Функция**
1	5 - 4 - 1 - 2 - 3	32
2	3 - 4 - 1 - 5 - 2	33
3	5 - 3 - 1 - 4 - 2	31
4	3 - 4 - 1 - 2 - 5	31

Ср. значение: $\frac{32 + 33 + 31 + 31}{4} = 31.75$

Поколение не изменилось.

III. Третье поколение:

1) Для первого скрещивания возьмем наборы последовательностей 1 и 2.

Возьмем первую границу после 3 числа, вторую после 5.

Получаем: |23| и |52|.

Аналогично прошлому поколению найдем потомков.

Для первого:

Получаем: 3 - 4 - 1 - 5 - 2 = 33

Для второго:

Получаем: 4 - 1 - 5 - 2 - 3 = 33

2) Теперь проведем скрещивание для последовательностей 3 и 4.

Возьмем границу после 1 и 3.

Получаем: |31| и |41|.

Аналогично прошлому поколению найдем потомков.

Для первого:

Получаем: 2 - 4 - 1 - 5 - 3 = 33

Для второго:

Получаем: 2 - 3 - 1 - 5 - 4 = 35

Проведем мутацию (если необходимо):

0.65, 0.92, 0.43, 0.16 — все значения больше порога мутации ⇒ мутацию не проводим.

Получаем:

№	Родители	Потомки	Функция
1	5 - 4 - 1 - 2 - 3	3 - 4 - 1 - 5 - 2	33
2	3 - 4 - 1 - 5 - 2	4 - 1 - 5 - 2 - 3	33
3	5 - 3 - 1 - 4 - 2	2 - 4 - 1 - 5 - 3	33
4	3 - 4 - 1 - 2 - 5	2 - 3 - 1 - 5 - 4	35

Уберем худшие особи.

Текущая популяция:

№	Последовательность	Функция
1	5 - 4 - 1 - 2 - 3	32
2	3 - 4 - 1 - 5 - 2	33
3	5 - 3 - 1 - 4 - 2	31
4	3 - 4 - 1 - 2 - 5	31

Ср. значение: $\frac{32 + 33 + 31 + 31}{4} = 31.75$

Поколение не изменилось. Заметим, что в популяции есть 2 оптимальные особи (3 и 4). Также отметим, что ни в одной из итераций генетического алгоритма не произошло мутаций.

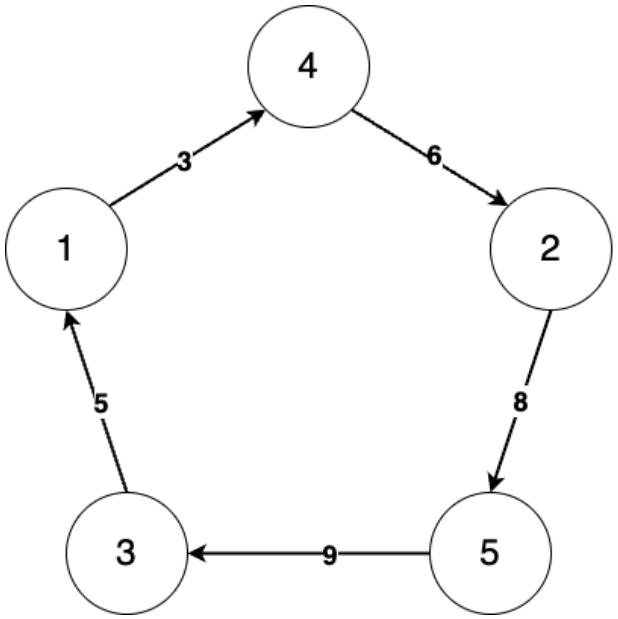
0, 4, 5, 3, 8
4, 0, 7, 6, 8
5, 7, 0, 7, 9
3, 6, 7, 0, 9
8, 8, 9, 9, 0

Вывод:

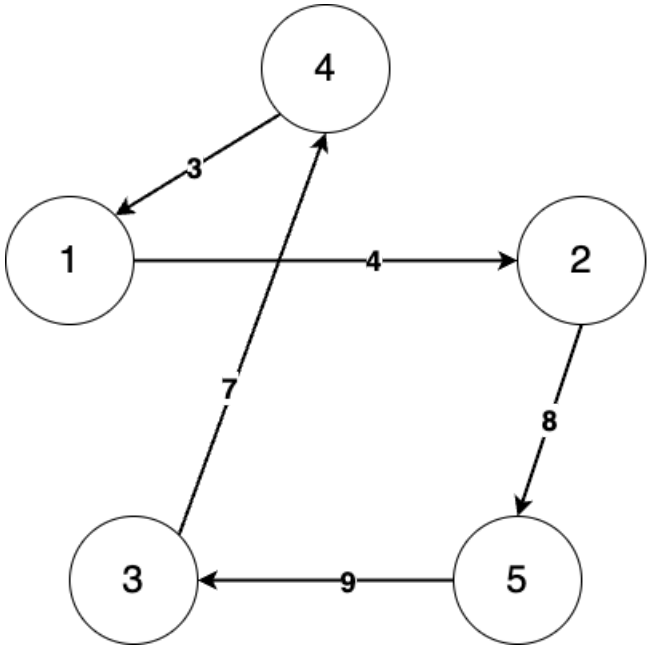
Я смог найти оптимальную последовательность при помощи генетического алгоритма, тем самым снизив среднее значение функции поколения.

Таким образом были получены следующие маршруты:

5 - 3 - 1 - 4 - 2:



3 - 4 - 1 - 2 - 5:



Код:

```
import math
import random

class SimulatedAnnealing:
  def __init__(self, temp, alpha, stopping_temp, stopping_iter, dist_matrix):
    self.sample_size = len(dist_matrix)
    self.temp = temp
    self.alpha = alpha
    self.stopping_temp = stopping_temp
    self.stopping_iter = stopping_iter
    self.iteration = 1
    self.dist_matrix = dist_matrix
    self.curr_solution = [i for i in range(0, len(dist_matrix))]
    self.best_solution = self.curr_solution
    self.solution_history = [self.curr_solution]
    self.curr_weight = self.weight(self.curr_solution)
    self.initial_weight = self.curr_weight
    self.min_weight = self.curr_weight
    self.weight_list = [self.curr_weight]

  def weight(self, sol):
    return sum([self.dist_matrix[i][j] for i, j in zip(sol, sol[1:] + [sol[0]])])

  def acceptance_probability(self, candidate_weight):
    return math.exp(-abs(candidate_weight - self.curr_weight) / self.temp)

  def accept(self, candidate):
    candidate_weight = self.weight(candidate)
    if candidate_weight < self.curr_weight:
      self.curr_weight = candidate_weight
      self.curr_solution = candidate
      if candidate_weight < self.min_weight:
        self.min_weight = candidate_weight
        self.best_solution = candidate
    elif random.random() < self.acceptance_probability(candidate_weight):
      self.curr_weight = candidate_weight
      self.curr_solution = candidate

  def anneal(self):
    while self.temp >= self.stopping_temp and self.iteration < self.stopping_iter:
      candidate = list(self.curr_solution)
      l = random.randint(2, self.sample_size - 1)
      i = random.randint(0, self.sample_size - l)
      candidate[i : (i + l)] = reversed(candidate[i : (i + l)])
      self.accept(candidate)
      self.temp *= self.alpha
```

```
            self.iteration += 1
            self.weight_list.append(self.curr_weight)
            self.solution_history.append(self.curr_solution)

def run_algorithm(params, matrix):
    temp, alpha, stopping_temp, stopping_iter = params
    sa = SimulatedAnnealing(temp, alpha, stopping_temp, stopping_iter, matrix.values.tolist())
    sa.anneal()
    best_fitness = sa.min_weight
    params = params
    logbook = sa.weight_list
    best_path = sa.best_solution
    return best_fitness, params, logbook, best_path

if __name__ == '__main__':
    params = input("input params: ")
    matrix = input("input matrix: ")
    run_algorithm(params, matrix)
```